

---

# Ultra-Compact Neuroprocessor for Automotive Diagnostics and Control

---

**R. Tawel and N. Aranki**  
Jet Propulsion Laboratory  
Pasadena, CA 91109

**L. A. Feldkamp and K. A. Marko**  
Ford Research Laboratory  
Dearborn, MI 48121-2053

## Abstract

Demands on the performance of vehicle control and diagnostic systems are steadily increasing as a consequence of stiff global competition and government mandates. In the United States, light trucks and passenger cars are required both to meet strict emission standards and to perform continuous diagnostics of all emissions systems operating in the vehicle. These requirements will become more comprehensive and difficult to achieve as emission standards are tightened in the next decade. Neural networks provide a means of creating control and diagnostic strategies that will permit these challenges to be met efficiently and robustly. Accordingly, a collaborative effort resulted in the development of a fast, low-cost custom ASIC capable of executing in real-time promising neural architectures. This paper describes this resulting VLSI design and the challenging application, misfire detection, that has served as a focus for the effort.

## 1 Introduction

The control system of a modern automobile involves several interacting subsystems, almost any one of which provides interesting theoretical and engineering challenges. Responsibility for many important functions has shifted from mechanical control to computer control. This has resulted in a substantial increase in fuel efficiency and, in combination with advances in catalyst technology, to a remarkable reduction in emissions (99% in a properly functioning system). This has been accomplished in the context of an extremely cost competitive industry. Increasingly stringent emissions regulations require that any malfunctioning component or system with the potential to undermine the emissions control system to be identified. Hence, cost effective ways of meeting these demands are being explored.

Neural networks have the potential for major impact in this work. Benefits may be anticipated in terms of the time required to design and calibrate a control or diagnostic strategy or in the observed performance. In particular, we have been investigating the use of neural networks as control structures (Puskorius and Feldkamp, 1996a), models, virtual sensors, and for misfire diagnostics (Puskorius and Feldkamp, 1996b, Feldkamp et al., 1996, Marko et al., 1996a,b,c). In most cases,

the computation performed by a neural network may be expected to exceed that of the strategy that it replaces. Though this may be justified by increased performance, a limit is set by processing power that remains after other powertrain control functions are handled. Though we have been able to execute neural networks for certain functions, such as idle speed control, directly in the engine processor, we anticipated that a broad realization of neural network potential might require specialized computational capability.

In the remainder of this paper we first provide a brief description of misfire detection, which is the most computationally challenging vehicle application we have encountered. Then we discuss aspects of using a recurrent neural network as a direct fault classifier for this problem. Finally, we describe the design requirements and constraints for VLSI to execute (not train) such networks and present the results of a comparison of software and hardware calculations for vehicle data.

## 2 The Misfire Diagnostic Problem

In order to perform effectively on board a vehicle, diagnostic algorithms must be extremely accurate and efficient. Detection of engine misfire is a particularly challenging problem, because the algorithm must diagnose approximately one billion events over the life of each vehicle, and perform that task between engine cylinder firings (which can occur at rates as high as 30,000 events per minute) without disturbing the computations required to carry out the control strategy.

Engine misfire is known to cause significant increases in tailpipe emissions and therefore has come under scrutiny as a major contributor to emissions problems which could be avoided through prompt failure detection and fault isolation on board the vehicle. While there are many ways one might try to diagnose engine misfire, the methods available today must rely on information from sensors already in use on production systems which have proven their accuracy and durability. This restriction limits the practical methods of misfire diagnostics to analysis of the engine crankshaft dynamics, observed with a crankshaft position sensor located at one end of the crankshaft. Basically, the strategy is to attempt to detect an crankshaft acceleration deficit following a cylinder firing and determine if that deficit is attributable to a lack of power provided on the most recent firing stroke.

The problem of detecting the acceleration deficit is complicated by several factors: 1) the crankshaft dynamics are influenced by unregulated inputs from the driver; 2) additional disturbances are introduced through the driveshaft from irregularities in the road; 3) the dynamics are obscured by measurement noise and process noise; 4) the diagnostics must run in real-time between engine firing events; and 5) the crankshaft is not infinitely stiff and exhibits complex dynamics which mask the signature of the misfire event and which are influenced by the event itself. In effect, we are observing the torsional oscillations of a nonlinear oscillator with driving forces applied at several locations along its main axis.

Figure 1a illustrates cylinder-by-cylinder crankshaft accelerations, taken when the engine is at high speed and lightly loaded. Acceleration deficits corresponding to misfire (here artificially induced) are not easy to spot. The task is to infer from such observations whether the driving forces produced by the engine combustion events correspond to normal combustion or engine misfire. While it is straightforward to write down the dynamical equations that approximate the crankshaft rotational dynamics as a function of the combustion pressures applied to the piston faces, it is quite difficult to solve those equations and extremely difficult to solve the inverse inference problem associated with misfire diagnostics. Nonetheless, the expectation of a discoverable dynamic relationship between the observed accelerations and the driving forces in this system, coupled with the absence of a satisfactory alternative approach, prompted our exploration of recurrent networks as a solution to the problem.

## 3 Time-Lagged Recurrent Networks

Recurrent networks with internal or external feedback have proven to be very useful as models or controllers for dynamical systems. Here we use a recurrent multi-layer perceptron (RMLP), which may be regarded as a combined generalization of a feedforward MLP and a one-layer fully recurrent network. Such networks are rather general approximators of dynamical systems. We have explored several ways of employing RMLPs for this problem; here we discuss their use as a direct classifier.

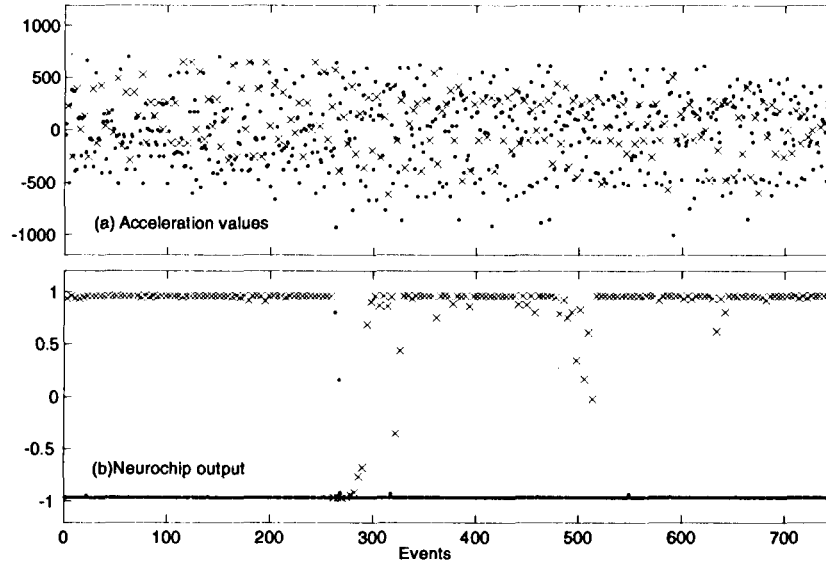


Figure 1: a) Temporal stream of acceleration values, illustrating the effects of crankshaft dynamics. Misfires are denoted by symbols 'x'. In the absence of torsional oscillations, the misfires would lie clearly below 0 on the vertical scale; b) Corresponding network outputs.

### 3.1 Network Architecture

The results to be shown below made use of the network architecture 4-15R-7R-1, i.e., 4 inputs, two fully recurrent hidden layers with 15 and 7 nodes, respectively, and a single output node. The activation function of each of the 23 computational nodes is a bipolar sigmoid. The network executes once per cylinder event (e.g., 8 times per engine cycle for an 8-cylinder engine). The inputs at time step  $k$  are the crankshaft acceleration (ACCEL, averaged over the last 90 degrees of crankshaft rotation), engine load (LOAD, computed from the mass flow of air), the engine speed (RPM), and a cylinder identification signal (CID, e.g., 1 for cylinder 1, 0 otherwise), which allows the network to synchronize with the engine cylinder firing order. This network contains 469 weights; thus one execution of the network requires 469 multiply-accumulate (MAC) operations and 23 evaluations of the activation function. It is this computational load ( $187,000 \text{ MAC s}^{-1}$ ) that is impractical in an already heavily loaded existing processor.

### 3.2 Training

Training recurrent networks often poses practical difficulties, primary among which is dealing with the *recency effect*, i.e., the tendency of a learning network to favor recent training examples at the expense of those previously encountered. To mitigate this difficulty we devised the *multi-stream training* technique (Feldkamp and Puskorius, 1994). This technique is especially effective in conjunction when weight updates are performed using the extended Kalman filter method (Singhal and Wu, 1989, Puskorius and Feldkamp, 1994). Experiments suggest that the present results would not easily have been obtained with the methods more commonly available.

The database used for network training was acquired by operating a test vehicle over as wide a range of operation as practically possible, including engine speed-load combinations that would rarely be encountered in normal driving. Misfire events are deliberately introduced (typically by interrupting the spark) at both regular and irregular intervals. A misfire alters the torsional oscillation pattern for several subsequent time steps, so it is important to provide the network with a range of misfire interval for all combinations of speed and load that correspond to positive engine torque. Though in this case the data used for training consists of more than 600,000 examples (one per cylinder event), it is clearly possible only to approximate complete coverage. Hence it is important to carry out extensive generalization testing and analysis.

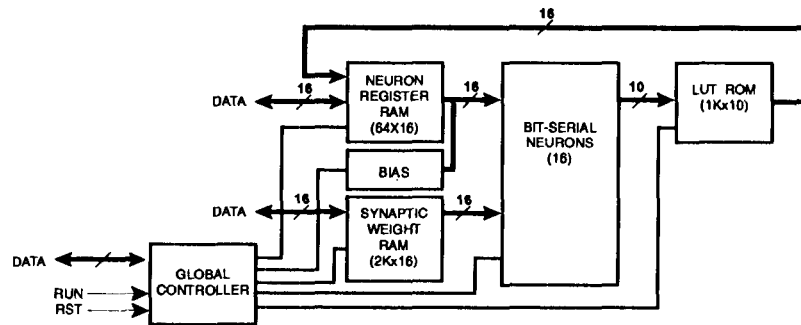


Figure 2: Schematic representation of forward propagation module.

Figure 1b shows the output of the trained RMLP for the acceleration data of Figure 1a. This data segment was not used in training the network (it was acquired *after* the network had been trained, thereby providing a modest test of network generalization). Though at present we have at our disposal only one test vehicle with this particular powertrain configuration, earlier work with a pair of test vehicles suggests that generalization among vehicles with nominally identical powertrains is entirely feasible.

## 4 Neuroprocessor Chip

An ever increasing number of diagnostic and control applications are being solved for the first time by the application of trainable neural classifiers of suitable capacity. These classifiers, however, are based upon systems which require considerable computational resources and as such must be implemented in dedicated silicon in order to meet the real-time computational requirements for both on-board diagnostics and control.

The *a priori* design constraints set forth at the onset of this collaborative effort called for the development of an **inexpensive, fully autonomous, and commercially viable** electronic chip. This single chip implementation was required to (1) be extremely compact in size (because of the mass market potential) (2) be flexible (so as to enable a number of different neural based applications to share the hardware and sequentially execute on it), and (3) offer high computational resolution (in order to avoid fixed-point induced arithmetic hardware diagnostic miscalls). By observing that even at red line internal combustion events occur on a millisecond time scale, a novel and extremely compact and powerful layer-multiplexed bit-serial neuromorphic architecture was developed and exploited so as to implement the recurrent neuromorphic formalism in custom CMOS silicon. It is this implementation that is discussed in the next sections along with a performance summary.

### 4.1 Architecture

At its most elemental level, neuromorphic computations can be summarized as a series of parallel multiply and accumulate operations interspersed by an occasional non-linear operation. In view of the driving constraints outlined in the previous section, we fully exploited five basic techniques to achieve our desired goals. These included usage of a (1) parallel intra-layer topology organized in a (2) single-instruction-multiple-data (SIMD) architecture. This architecture made full use of both (3) bit-serial fixed-point computational techniques; and (4) inter-layer multiplexing of neuron resources. Lastly (5) nonlinearities were handled by the use of look-up-tables.

This resulting architecture is shown schematically in Figure 2 and consists of: (1) a global controller; (2) a pool of 16 bit-serial neurons; (3) a bipolar sigmoid activation ROM look-up-table; (4) neuron state registers; and (5) a synaptic weight RAM. In this design, both inputs to the network as well as neuronal outputs are stored in the neuron state RAM. When triggered by the global controller, each of the 16 neurons performs the neuronal multiply and accumulate (MAC) operation. They receive as input the synaptic weights (from the synaptic weight RAM) and activations from either (a) input nodes or (b) neurons on a previous layer in a bit serial fashion, and output the accumulated sum of partial products onto a tri-stated bus which is commonly shared by all 16 neurons. Because of

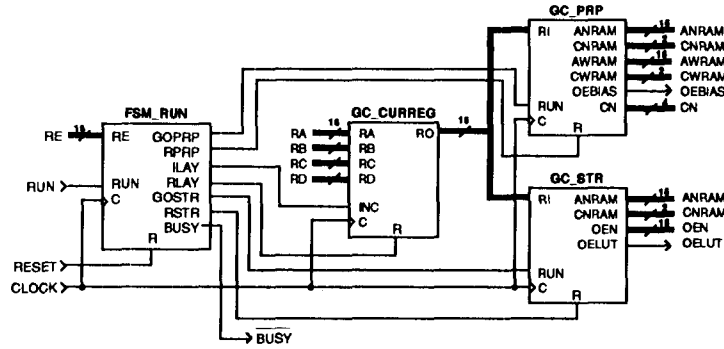


Figure 3: Run-time forward propagation controller.

the computational nature of neural networks - where information is sequentially computed a layer at a time - only enough neurons are physically implemented in silicon as exist on the layer with the largest number of neurons for all applications of interest. As such, a candidate pool of 16 silicon neurons was chosen. This means that the number of "real" or non-recurrent neurons on any given layer is bounded by [1, 16]. This intra-layer pool of neurons is organized in a SIMD configuration. By single-instruction (SI) we mean that all active neurons in the pool execute the same instruction at the same time. Multiple-data (MD) means that each active neuron acts on its own slice of data, independently of all other processors. Together this means that at the intra-layer level, the chip performs fully parallel computations under the control of the global controller.

A significant reduction in silicon real-estate was achieved by performing inter-layer multiplexing of the 16 neuron pool. Inter-layer multiplexing refers to reusing the hardware used in calculating the activations of neurons in one layer for the calculation of neurons in another layer. Since neurocomputations are performed a layer at a time, this reuse of hardware is aimed at increasing the utilization of the hardware that would otherwise remain idle. This re-utilization of hardware leads to a significant reduction of the required VLSI real estate. The general idea behind layer-multiplexing is to reuse the circuitry dedicated to one layer during the evaluation of the next layer. In this way, only enough hardware to accommodate the layer with the largest number of neurons needs to be physically incorporated in hardware. Other smaller layers can then reuse portions of this hardware during their evaluation.

Bit-serial algorithms for arithmetic operations are most suitable for efficient VLSI implementations because of their canonical nature and minimal interconnection requirements. For this reason, we made extensive use of bit-serial techniques to enable us to incorporate onto a single compact chip a complete stand-alone neuroprocessor.

## 4.2 Controller

At the heart of the neuroprocessors architecture is the global controller. The controller contains the logic to enable the neurochip to execute its task. This task is to load an architecture from RAM, and once triggered, to generate all necessary control signals in addition to orchestrate data movement on-chip and off-chip. When there are no computations being performed, the global controller remains in the idle state, signalling its availability by having the active low BUSY flag set high. When a LOAD command is issued, the controller reads from RAM a neural network topology and goes into an idle state. When the RUN command is subsequently issued, the global controller is in charge of providing control signals to the 16 on-chip neurons, the RAM and the ROM in order to proceed with the desired neurocomputation. Input activations are read out of the 64x16 Neuron State RAM, synaptic weights are read out of the 2Kx16 Synaptic Weight RAM, and both are propagated to the bank of 16 neurons. In this way, the global controller keeps track of both intra-layer operations as well as inter-layer operations. Upon completion of a forward pass through the network architecture, the global controller asserts the BUSY flag and returns to the idle state.

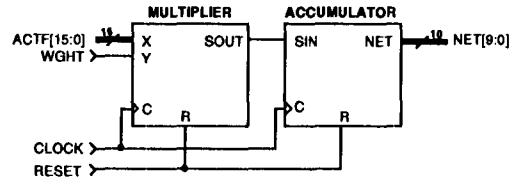


Figure 4: Bit-serial neuron.

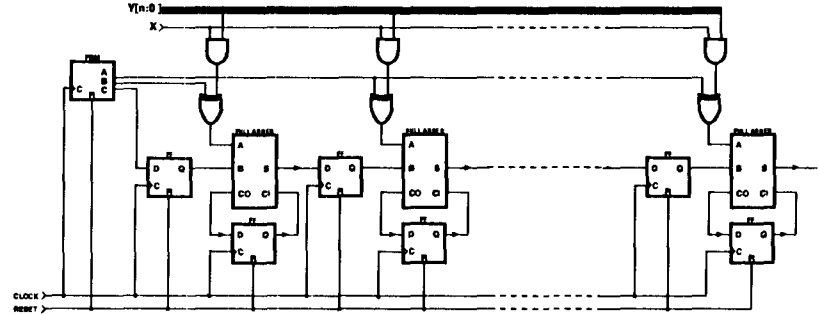


Figure 5: Bit-serial multiplier of length  $n$ .

### 4.3 Neurons

Fixed-point bit-serial algorithms for operations such as addition and multiplication are uniquely suitable for efficient VLSI implementations because of their highly compact representations. For example, the size of an  $n \times n$  bit multiplier scales quadratically ( $O(n^2)$ ) for a bit-parallel implementation and linearly ( $O(n)$ ) for a bit-serial one. Such serial based computational techniques were therefore exploited in the design of the neurons. A schematic representation of a bit-serial neuron is shown in Figure 4. Each of the 16 neurons in the chip architecture is a duplicate of the same structure.

The multiplier shown in Figure 4, is used to perform the synaptic multiplications. The driving precision constraints for the misfire problem called for the use of a 16x16 bit fixed-point multiplier. In operation, the multiplier accepts as input either an input stimulus to the neural network, or an activation output from a neuron on a previous layer. It multiplies this quantity by the corresponding synaptic weight. The input stimulus (or activation output) is presented to the multiplier in a bit-parallel fashion, while the synaptic weights are presented in a bit-serial fashion. The serial output of the multiplier feeds directly into an accumulator.

The multiplier shown in Figure 5, is a modified and improved version of previously reported serial multiplier. Any size multiplier can be formed by cascading the basic multiplier cell. The bit-wise multiplication of the multiplier and multiplicand is performed by the AND gates. At each clock cycle, the bank of AND gates compute the partial product terms of the multiplier  $Y[15:0]$  and the serial multiplicand  $X(t)$ . Two's complement multiplication is achieved by using XOR gates on the outputs of the AND gates. By controlling one of the inputs of the XOR gate, the finite state machine FSM can form the two's complement of selected terms based on its control flow. In general, for an  $n \times n$  multiplier (resulting in a  $2n$  bit product), the multiplier can be formed by using  $2n$  basic cells and will perform the multiplication in  $2n + 2$  clock cycles. Successive operations can be pipelined and the latency of the LSB of the product is  $n + 2$  clock cycles.

The accumulator, shown in Figure 6, is also of a bit-serial design. It is extremely compact as it consists of a single bit-serial adder linked to a chain of data registers. The length of the accumulator chain is governed by the multiplication length. The multiplier takes  $2n + 2$  clock cycles to perform a complete  $n \times n$  multiplication. At each clock cycle, the accumulator sums the bit from the input data stream with both the current contents of the data register on the circular chain as well as any carry bits that might have been generated from the addition in the previous clock cycle. This value is subsequently stored onto the chain on the next clock cycle. This creates a circulating chain of

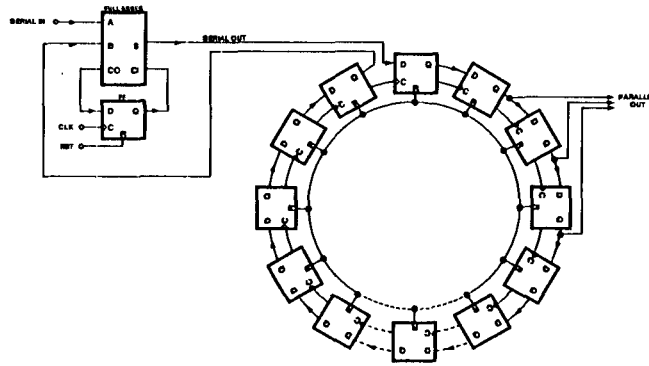


Figure 6: Bit-serial accumulator of length  $n$ .

data bits in the accumulator with period  $2n + 2$ .

## 5 Performance & Summary

A novel fixed-point bit-serial recurrent neuroprocessor chip has been designed and fabricated. This digital stand-alone ASIC was designed as a co-processor chip to the host engine computer CPU. It is capable of sequentially executing multiple neural based diagnostic and control applications between engine events. The chip was successfully deployed and field tested on a number of diagnostic and control problems including – the engine misfire detection problem – in a *production Ford Grand Marquis* automobile.

The neuroprocessor design was implemented using HP's  $0.5\ \mu\text{m}$  CMOS design rules. The first generation chip measured  $8\text{mm}^2$  in size. For extreme design compactness, the chip made extensive use of a variety of design techniques; including intra-layer parallelism, inter-layer multiplexing, and fixed-point bit-serial based computational techniques. Flexibility of the design was achieved by allowing the architecture to be on-the-fly programmable from RAM. The intra-layer architecture of the neuroprocessor was organized in a SIMD configuration. To accommodate both bipolar sigmoids as well as excitatory and inhibitory synaptic weights, fixed point two's complement arithmetic was used.

The current design operates at a conservative 20 MHz clock speed. A neural application can be loaded into the hardware in under  $1\ \mu\text{s}$ . Because of the SIMD architecture, it takes  $1.6\ \mu\text{s}$  to simultaneously perform 16 multiply and accumulate operations. This translates into an effective computational throughput of  $0.1\ \mu\text{s}$  per MAC operation. For the engine misfire 4-15R-7R-1 topology, the entire diagnostic classification was performed in under  $80\ \mu\text{s}$ . The next generation processor will operate at 50 MHz with all timings scaling inversely proportionally.

## 6 Acknowledgements

The research described in this paper was performed by the Center for Space Microelectronics Technology, Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the National Aeronautics and Space Administration, Office of Space Science.

## 7 References

- Feldkamp, L. A. and G. V. Puskorius (1994) Training controllers for robustness: Multi-stream DEKF. *Proceedings of the IEEE International Conference on Neural Networks*, Orlando, pp. 2377–2382.
- Feldkamp, L. A., G. V. Puskorius, K. A. Marko, J. V. James, T. M. Feldkamp, and G. Jesion (1996) Unravelling dynamics with recurrent networks: Application to engine diagnostics. *Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems*, New Haven, CT, pp. 59–64.

- Marko, K. A., J. V. James, T. M. Feldkamp, G. V. Puskorius, and L. A. Feldkamp (1996a) Applications of Neural Networks to the Construction of "Virtual" Sensors and Model-Based Diagnostics. *Proceedings of 29th International Symposium on Automotive Technology and Automations (ISATA)*, Florence, Italy, pp. 133-138.
- Marko, K. A., J. V. James, T. M. Feldkamp, G. V. Puskorius, and L. A. Feldkamp. (1996b) Signal Processing by Neural Networks to Create "Virtual" Sensors and Model-Based Diagnostics. *Proceedings of the International Conference on Artificial Neural Networks (ICANN'96)*, Bochum, Germany, pp. 191-196.
- Marko, K. A., J. V. James, T. M. Feldkamp, G. V. Puskorius, L. A. Feldkamp, and D. Prokhorov (1996c) Training Recurrent Networks for Classification: Realization of Automotive Engine Diagnostics. *Proceedings of the World Congress on Neural Networks (WCNN'96)*, San Diego, CA, pp. 845-850.
- Puskorius, G. V. and L. A. Feldkamp (1994) Neurocontrol of Nonlinear Dynamical Systems with Kalman Filter-Trained Recurrent Networks. *IEEE Transactions on Neural Networks* 5, pp. 279-297.
- Puskorius, G. V., L. A. Feldkamp, and L. I. Davis, Jr. (1996) Dynamic neural network methods applied to on-vehicle idle speed control. *Proceedings of the IEEE*, vol. 84, no. 10, pp. 1407-1420.
- Puskorius, G. V. and L. A. Feldkamp (1996) Signal processing by dynamic neural networks with application to automotive misfire detection. *Proceedings of the 1996 World Congress on Neural Networks*, San Diego, pp. 585-590.
- Singhal, S. and L. Wu (1989) Training Multilayer Perceptrons with the Extended Kalman algorithm, In D. S. Touretzky (ed) *Advances in Neural Information Processing Systems 1*, pp. 133-140. San Mateo, CA: Morgan Kaufmann.